

ABSTRACT DATATYPES

- An abstract datatype is a collection of data and set of operations on that data

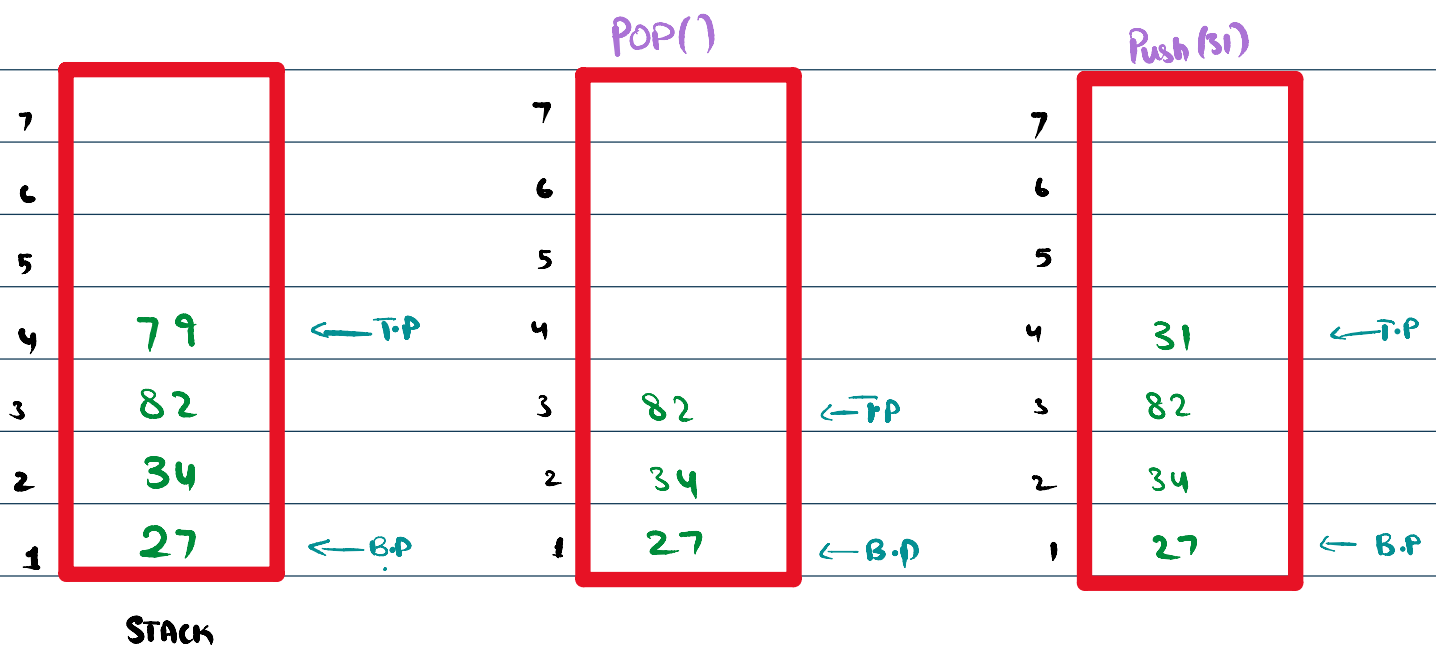
① STACKS

② QUEUES

③ LINKED LIST

STACKS

- A list containing several items operating on the Last in first out (LIFO) principle. Items can be added to the stack (Push) and removed from the stack (POP). The first item added to the stack is the last item removed from stack



B.P = Base Pointer

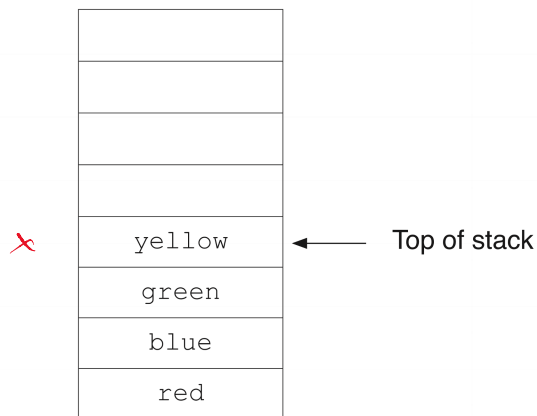
B.P = Base Pointer

T.P = Top Pointer

- A stack can be implemented using an array and set of pointers.

2

- 1 (a) A stack contains the values 'red', 'blue', 'green' and 'yellow'.



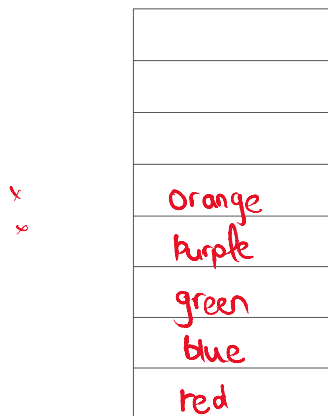
- (i) Show the contents of the stack in **part(a)** after the following operations.

ALWAYS
FROM
TOP

← POP ()

PUSH ('purple')

PUSH ('orange')



3

(ii) Show the contents of the stack from **part(a)(i)** after these further operations.

POP ()

POP ()

PUSH ('brown')

✗ POP ()

PUSH ('black')

black
green
blue
red

[1]

(b) A queue is an alternative Abstract Data Type (ADT).

Describe a **queue**.

.....

.....

.....

.....

.....

..... [3]

8

- 3 Joseph is taking a toy apart. Each time he removes an item from the toy, he writes the name of the item at the bottom of a paper list. When he rebuilds the toy, he puts the items back together working from the bottom of the list.

Joseph writes a computer program to create the list using a stack, `Parts`.

- (a) Describe a stack structure.

An Abstract datatype that follows Last in First out

[1]

- (b) The stack is represented as an array in the program, the first element in the array is `[0]`.

The current contents of the stack, `Parts`, and its pointer, `StackPointer` are shown.

`StackPointer`

5

`StackContents`

0	"Screw 1"
1	"Screw 2"
2	"Back case"
3	"Screw 3"
4	"Engine outer"
5	
6	
7	

- (i) Describe the purpose of the variable `StackPointer`.

Points to the next free space on the stack

(i) Describe the purpose of the variable `StackPointer`.

Points to the next free space on the stack

[1]

© UCLES 2018

9608/41/O/N/18

9

(ii) The procedure `POP()` removes an item from the stack. The procedure `PUSH(<identifier>)` adds an item to the stack.

The current contents of the stack, `Parts`, and its pointer, `StackPointer` are shown.

`StackPointer`

5

`StackContents`

0	"Screw 1"
1	"Screw 2"
2	"Back case"
3	"Screw 3"
4	"Engine outer"
5	
6	
7	

x
x

L1
L2 x
w1 x

Use the table below to show the contents of the stack, `Parts`, and its pointer after the following code is run.

`POP()`

Use the table below to show the contents of the stack, FIELDS, and its pointer after the following code is run.

```
POP()
POP()
PUSH("Light 1")
PUSH("Light 2")
PUSH("Wheel 1")
POP()
POP()
```

StackPointer

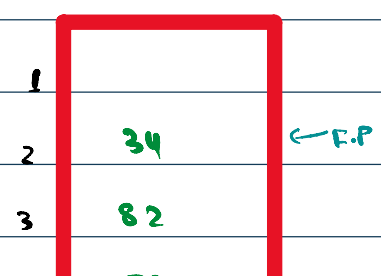
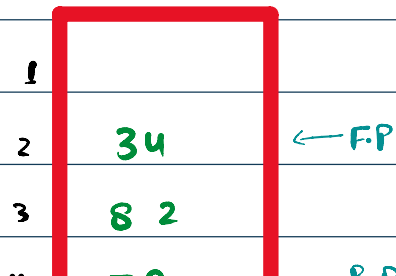
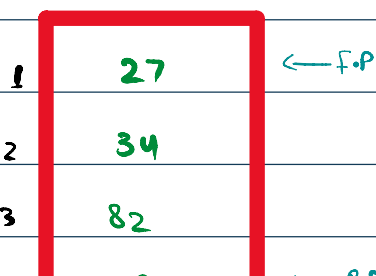
4

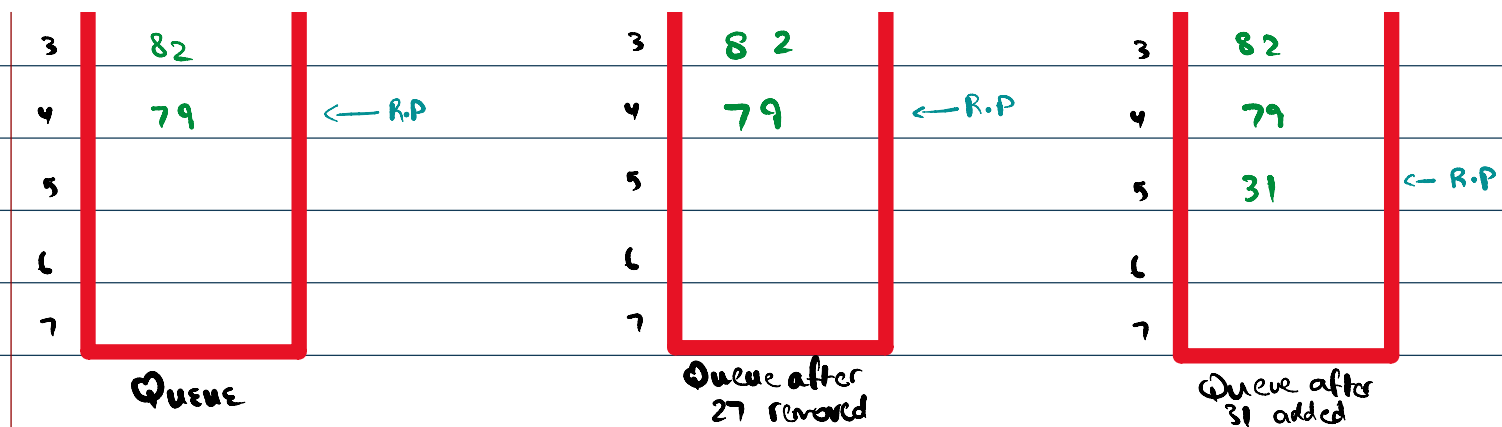
StackContents

0	'Screw 1'
1	'Screw 2'
2	'Black Cage'
3	'Light 1'
4	
5	
6	
7	

Queues

- A list containing several items operating on the (First in First Out) principle
- The first item added is the first item remove from the queue.





F.P : Front Pointer

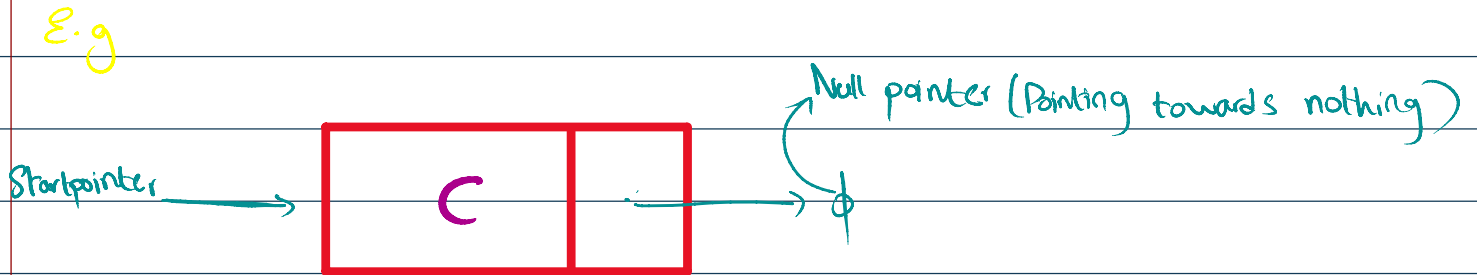
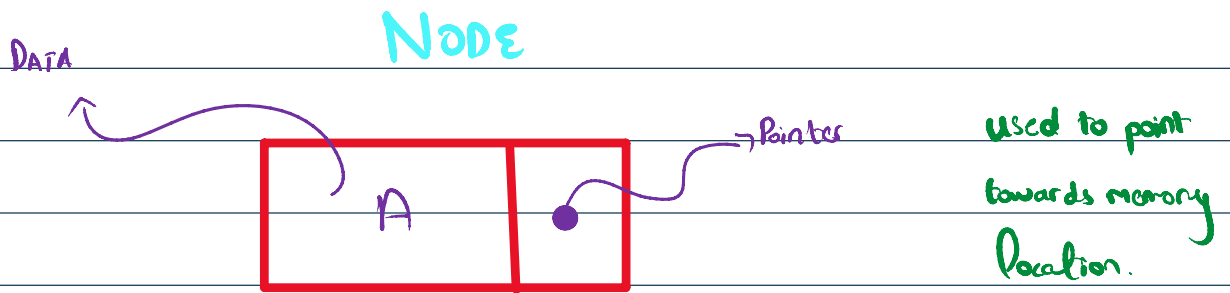
R.P : Rear Pointer

DESCRIBE A QUEUE

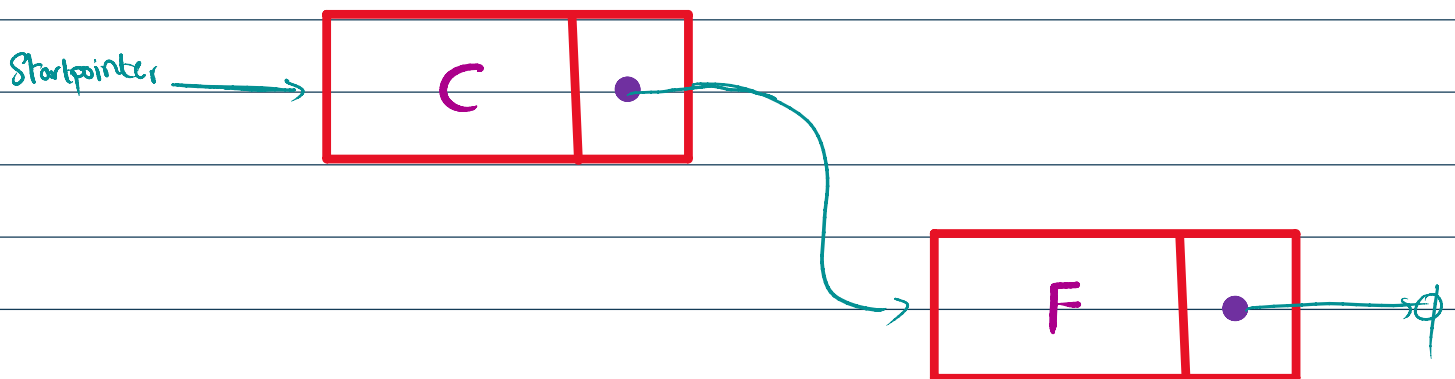
- Linear datastructure
- First in First out // An item is added to the end of the queue and an item is removed from the front
- All items are kept in the order they are entered
- It has a head pointer and a tail pointer.

LINKED LIST

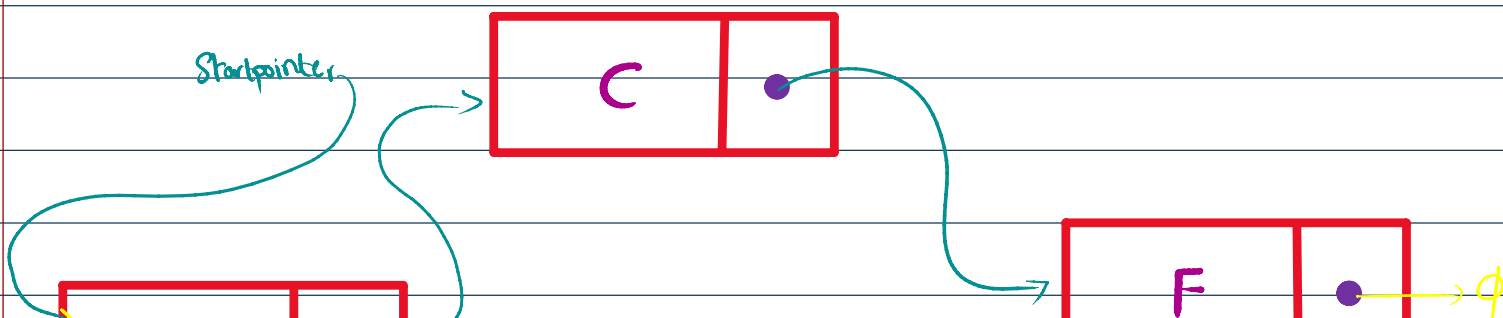
- A list containing several items in which each item in the list points to the next item in the list. In a linked list a new item is always added to the start of the list.

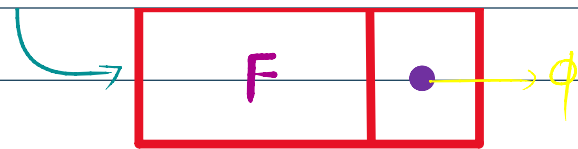
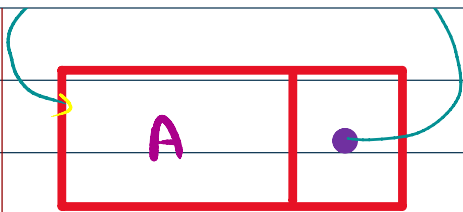


Now Connect One more Element F

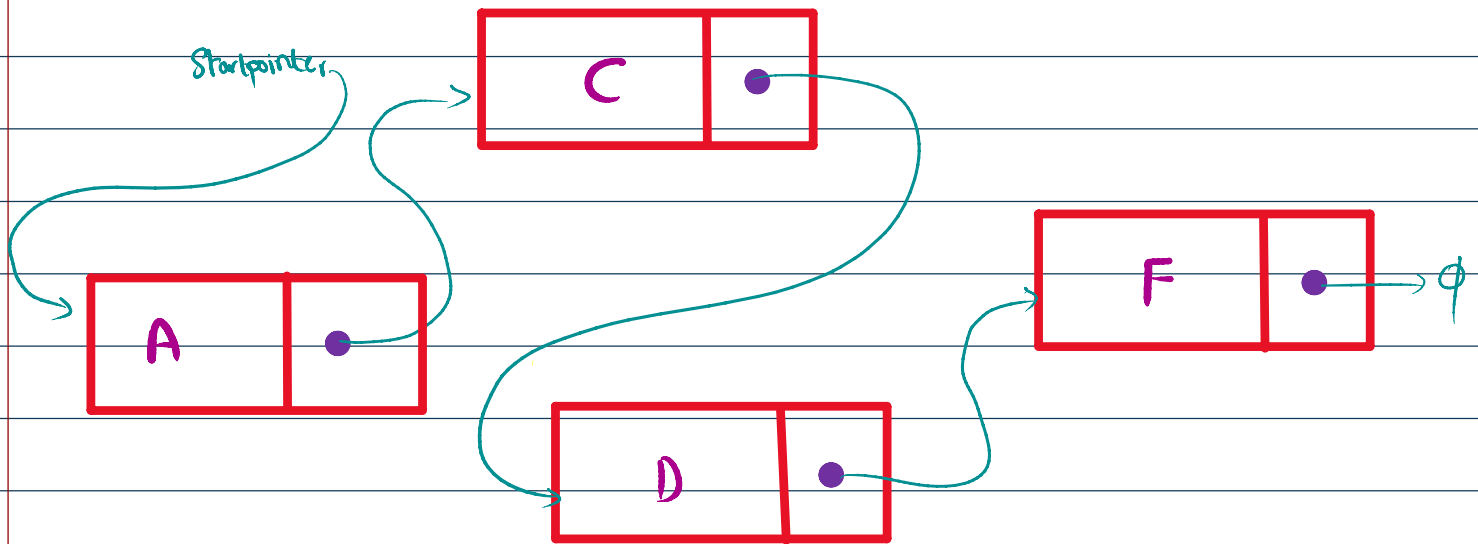


Now Connect One more Element A in start

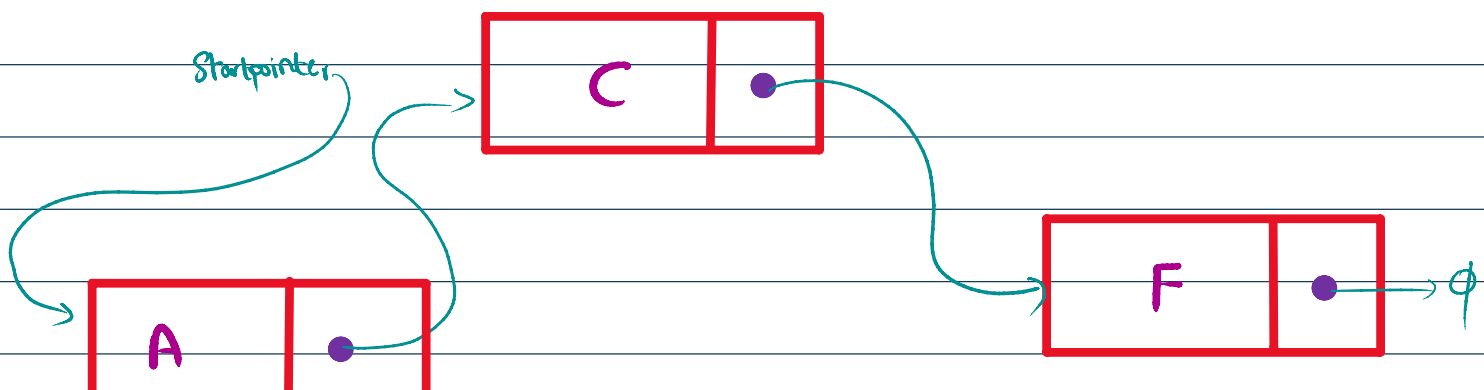


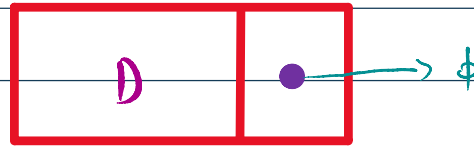
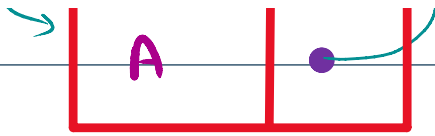


Now Connect One more Element D

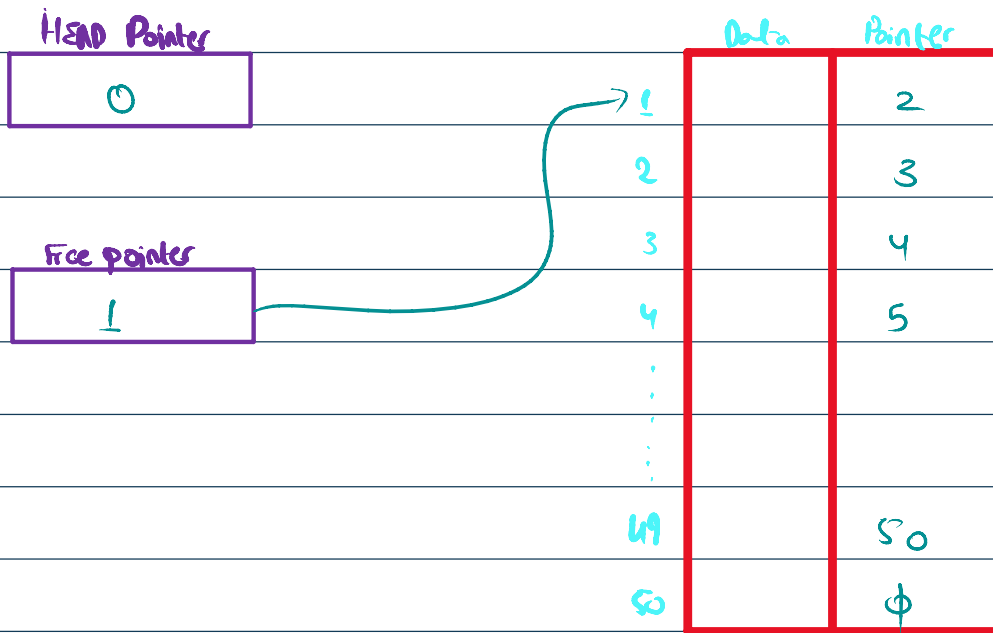


Now Remove Element D





INITIALIZING LINKED LIST IN ARRAY STRUCTURE



PASTPAPER QUESTION RELATED TO LINKED LIST

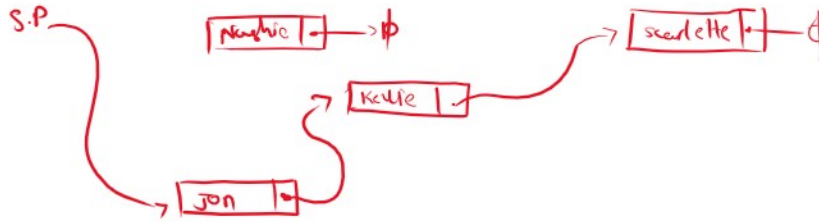
3 A linked list Abstract Data Type (ADT) has these associated operations.

1. Create linked list
2. Add item to linked list
3. Remove item from linked list

Each node in a linked list consists of data and a pointer to the next item in the linked list. Items in the linked list are maintained in order.

(a) A linked list is to be set up that stores names in alphabetical order. Show the final state of this linked list after the following operations are carried out.

```
CreateLinkedList
AddItem("Nushie")
AddItem("Kellie")
AddItem("Scarlett")
RemoveItem("Nushie")
AddItem("Jon")
```



[2]

(b) A programming language provides built-in array data structures. This linked list is to be implemented using these array data structures.

Define a record type, `ListNode`, for each node.

```

Type ListNode
  Declare Data: string
  Declare Pointer: Integer
EndType
  
```

[3]

(c) Write an array declaration to reserve space for 50 nodes in array `NameList`.

DECLARE NameList [1:50] OF ListNode [2]

(d) (i) The `CreateLinkedList` operation links all nodes to form the free list and initialises the `HeadPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers.

NameList		
	Name	Pointer
	HeadPointer	0
	FreePointer	1
[1]		2
[2]		3
[3]		4
[4]		5
:		
:		
[49]		50
[50]		0

[4]

LINKED LIST WITH STACK

10

5 A stack Abstract Data Type (ADT) has these associated operations:

- create stack
- add item to stack (push)
- remove item from stack (pop)

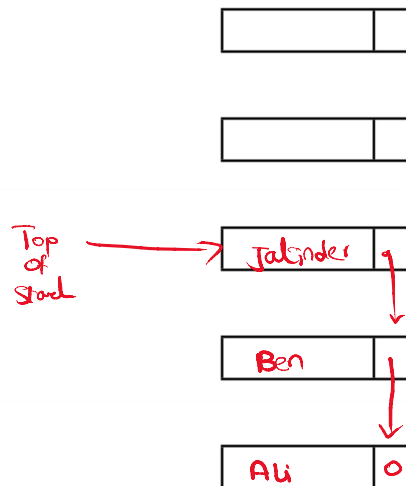
The stack ADT is to be implemented as a linked list of nodes.

Each node consists of data and a pointer to the next node.

(a) There is one pointer: the top of stack pointer, which points to the last item added to the stack. Draw a diagram to show the final state of the stack after the following operations are carried out.

```
CreateStack  
Push("Ali")  
Push("Jack")  
Pop  
Push("Ben")  
Push("Ahmed")  
Pop  
Push("Jatinder")
```

Add appropriate labels to the diagram to show the final state of the stack. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

(b) Using pseudocode, a record type, Node, is declared as follows:

```
TYPE Node
  DECLARE Name : STRING
  DECLARE Pointer : INTEGER
ENDTYPE
```

The statement

```
DECLARE Stack : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array Stack.

(i) The CreateStack operation links all nodes and initialises the TopOfStackPointer and FreePointer.

Complete the diagram to show the value of all pointers after CreateStack has been executed.

TopOfStackPointer		Stack	
		Name	Pointer
0 FreePointer 1	[1]		2
	[2]		3
	[3]		4
	[4]		5
	[5]		6
	[6]		7
	[7]		8
	[8]		9
	[9]		10
	[10]		0

[4]

LINKED LIST WITH QUEUE

6 A queue Abstract Data Type (ADT) has these associated operations:

- create queue
- add item to queue
- remove item from queue

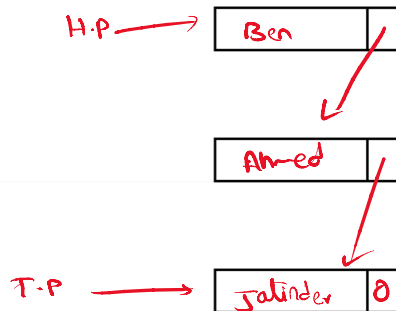
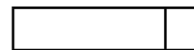
The queue ADT is to be implemented as a linked list of nodes.

Each node consists of data and a pointer to the next node.

(a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Add appropriate labels to the diagram to show the final state of the queue. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

(b) Using pseudocode, a record type, Node, is declared as follows:

```
TYPE Node
  DECLARE Name      : STRING
  DECLARE Pointer   : INTEGER
ENDTYPE
```

The statement

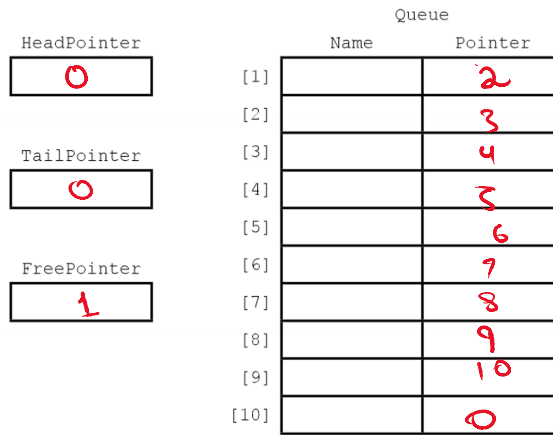
```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array Queue.

(i) The CreateQueue operation links all nodes and initialises the three pointers that need to be used: HeadPointer, TailPointer and FreePointer.

Complete the diagram to show the value of all pointers after CreateQueue has been executed.

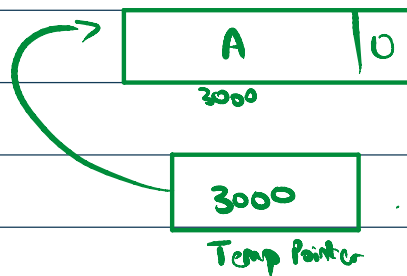
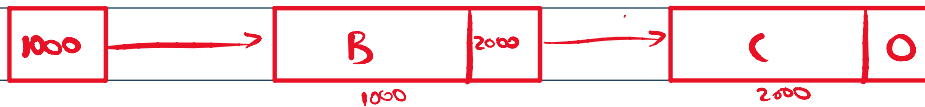
Complete the diagram to show the value of all pointers after CreateQueue has been executed.



[4]

INSERTION OF A NODE IN THE BEGINNING

Start Pointer



- (1) First we will connect pointer of A to pointer of B by using the address in start pointer
- (2) Then we will point the head pointer to B by using Temp pointer.

Why we can not point head pointer to B first?

Because after that we will not have another way to link pointer B to A the reference of point A will be lost.

INSERTION OF A NODE IN THE MIDDLE

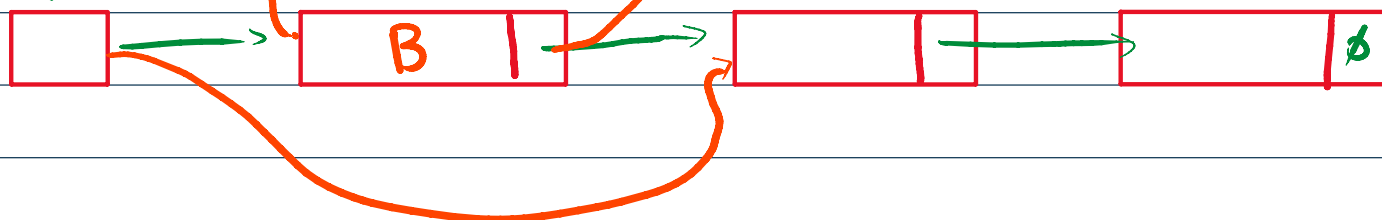
.NOTE : Use a Free List.

Question : Add data value B in between A and C

S.P



S.P



- ① Check for a free node in a linked list.
- ② Search for correct insertion point.
- ③ Assign data value B to the first node in free list

- ③ Assign data value B to the first node in free list
- ④ Pointer from A will be changed to point to node containing B
- ⑤ Pointer from B will be changed to point to node containing C
- ⑥ Start pointer in free list moved to point to next free node.

Question : how the linked list may be implemented by using arrays.

- Define a record data comprising a data element and a pointer and declare an array of this type
- An integer variable to hold the start pointer and an integer variable to store the next free pointer.